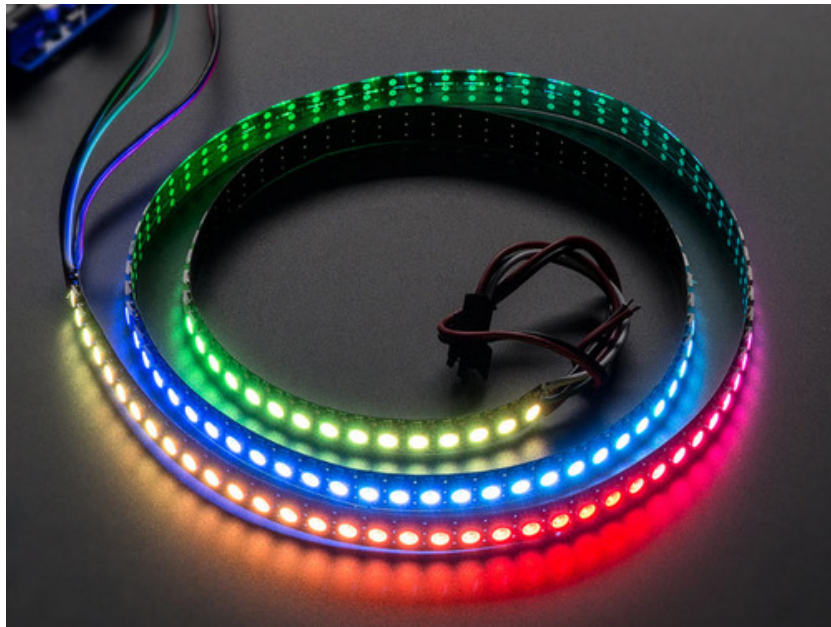




# Adafruit NeoPixel Überguide

Created by Phillip Burgess



Last updated on 2014-06-28 08:16:54 PM EDT

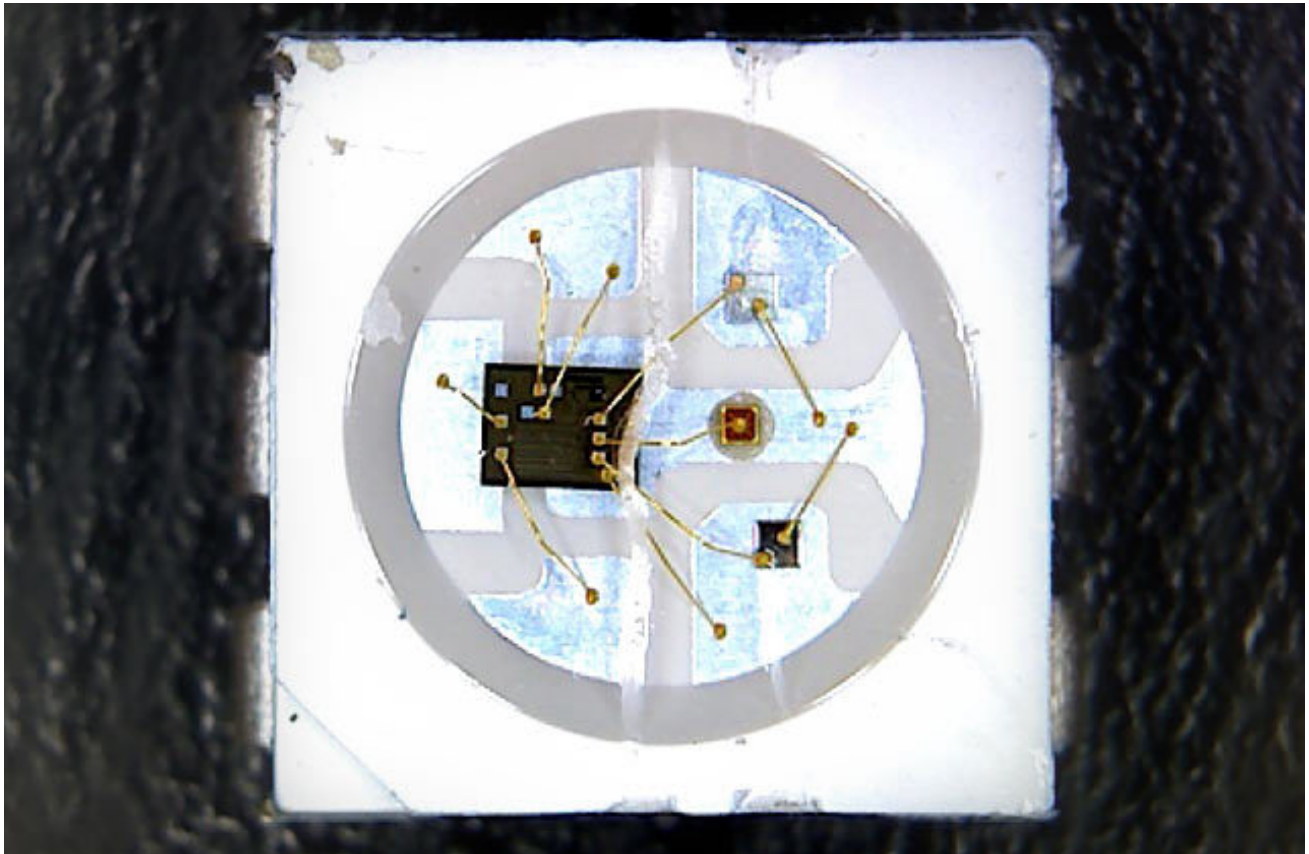
## Guide Contents

Guide Contents	2
The Magic of NeoPixels	3
Important Things to Know About NeoPixels in General	3
Adafruit NeoPixels are Available in the Following Products:	4
Important Things to Know About NeoPixel Strips	6
Important Things to Know About NeoPixel Rings	9
Best Practices	14
Improper use can damage your NeoPixels. Before diving in, be aware of the following:	
Arduino Library	16
Basic Connections	16
A Simple Code Example: strandtest	18
Pixels Gobble RAM	22
Powering NeoPixels	23
Estimating Power Requirements	25
Giant Power Supplies	26
Distributing Power	27
Driving 5V NeoPixels from 3.3V Microcontrollers	28
NeoMatrix Library	29
Layouts	30
Tiled Matrices	33
Other Layouts	35
RAM Again	35
Gamma Correction	35
Advanced Coding	37
Third-Party Libraries	38
Writing Your Own Library	39

# The Magic of NeoPixels

Incorporating scads of LEDs into an electronic project used to be a hairy prospect, a veritable rat's nest of wires and code. The arrival of dedicated LED driver chips brought welcome relief, offloading grunt work from the microcontroller and allowing one to focus on the application. Much simpler, but still not “Christmas light” simple.

The **WS2812** Integrated Light Source — or **NeoPixel** in Adafruit parlance — is the latest advance in the quest for a simple, scalable and affordable full-color LED. Red, green and blue LEDs are integrated alongside a driver chip into a tiny surface-mount package controlled through a single wire. They can be used individually, chained into longer strings or assembled into still more interesting form-factors.



We know you're eager to get started...but If this is your first time using NeoPixels, please at least read the “Best Practices” page before connecting anything!

## Important Things to Know About NeoPixels in General

- Not all addressable LEDs are NeoPixels. “NeoPixel” is Adafruit’s brand for individually-addressable RGB color pixels and strips based on the WS2812 and WS2811 LED/drivers, using a single-wire control protocol. Other LED products we carry — WS2801 pixels, LPD8806 and “analog” strips — use different methodologies (and have

their own tutorials). *When seeking technical support in the forums, a solution can be found more quickly if the correct LED type is mentioned.*

- NeoPixels don't just light up on their own; they require a microcontroller (such as Arduino) and some programming. We provide some sample code to get you started. To create your own effects and animation, you'll need some programming practice. If this is a new experience, work through some of the beginning Arduino tutorials to get a feel for the language.
- NeoPixels aren't the answer for every project. The control signal has very strict timing requirements, and some development boards (such as Netduino or Raspberry Pi) can't reliably achieve this. This is why we continue to offer other LED types; some are more adaptable to certain situations.

Can I use NeoPixels for POV (persistence of vision) displays?

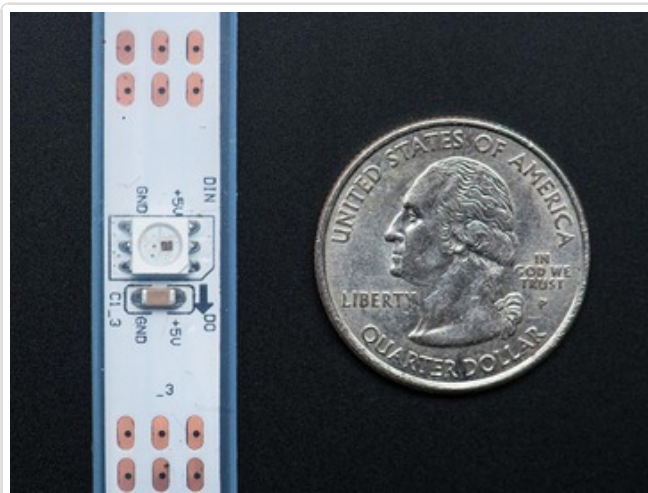
Not recommended. The refresh rate is relatively low (about 400 Hz), and color displays in fast motion may appear "speckled." They look fine in stationary displays though (signs, decorations, jewelry, etc.). For POV use, [LPD8806 strips \(http://adafru.it/306\)](http://adafru.it/306) will look much better (they have about a 4 KHz refresh rate).

How about for light painting?

Definitely! The slower movement used for photographic light painting doesn't call attention to the limited refresh rate; the results look great, especially with a light diffuser.

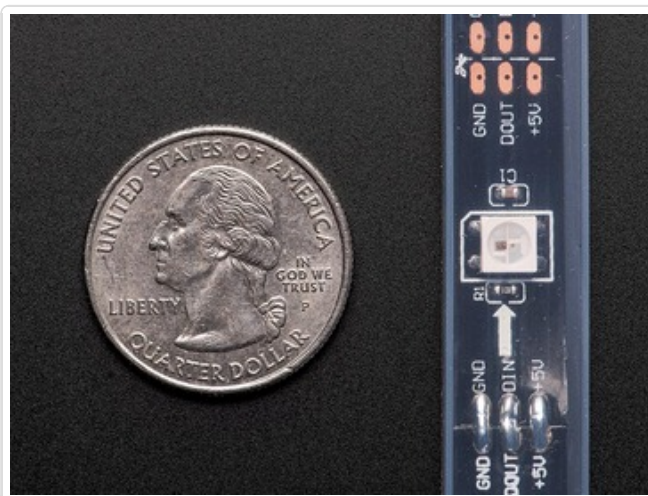
## Adafruit NeoPixels are Available in the Following Products:

---



**NeoPixel Digital RGB LED Weatherproof Strip** is available in three different “densities”: 30, 60 and 144 LEDs per meter, on a white or black backing strip.

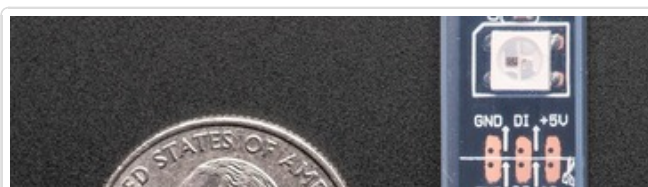
- **30 LEDs per meter, white strip** (<http://adafru.it/1376>)
- **30 LEDs per meter, black strip** (<http://adafru.it/1460>)
- **60 LEDs per meter, white strip** (<http://adafru.it/1138>)
- **60 LEDs per meter, black strip** (<http://adafru.it/1461>)
- **144 LEDs per meter, white strip** (<http://adafru.it/1507>)
- **144 LEDs per meter, black strip** (<http://adafru.it/1506>)



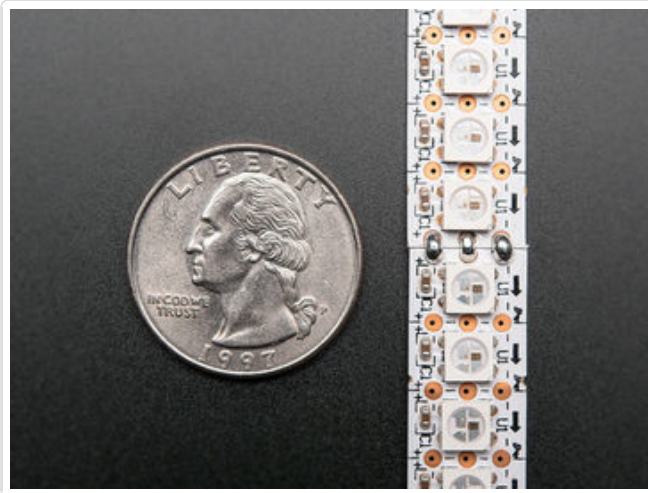
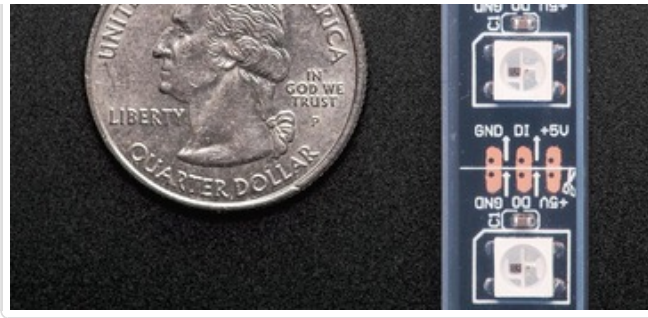
The *approximate* peak power use (all LEDs on at maximum brightness) per meter is:

- **30 LEDs:** 9.5 Watts (just under 2 Amps at 5 Volts).
- **60 LEDs:** 18 Watts (about 3.6 Amps at 5 Volts).
- **144 LEDs :** 35 watts (7 Amps at 5 Volts).

Mixed colors and lower brightness settings will use proportionally less power.





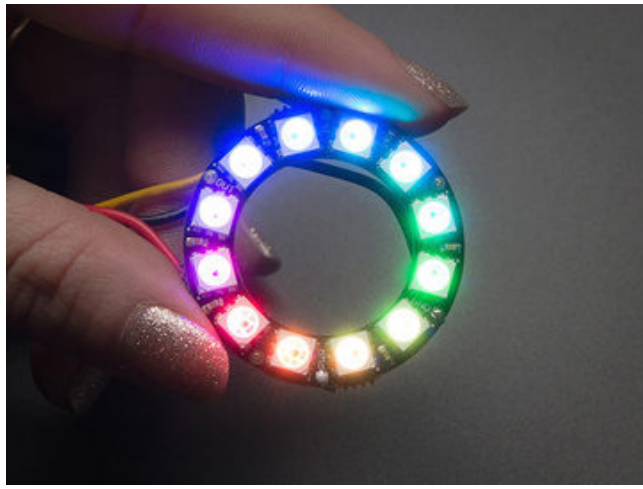


The product list continues below. But first...

## Important Things to Know About NeoPixel Strips

- NeoPixel strips are sold in one meter lengths. The 144 pixels/meter strips are always separate 1-meter lengths. For 60 and 30 pixels/meter strips, orders for multiple meters will be a single contiguous strip, up to a limit: 4 meters for 60 pixels/meter strip, or 5 meters for 30 pixels/meter strip.

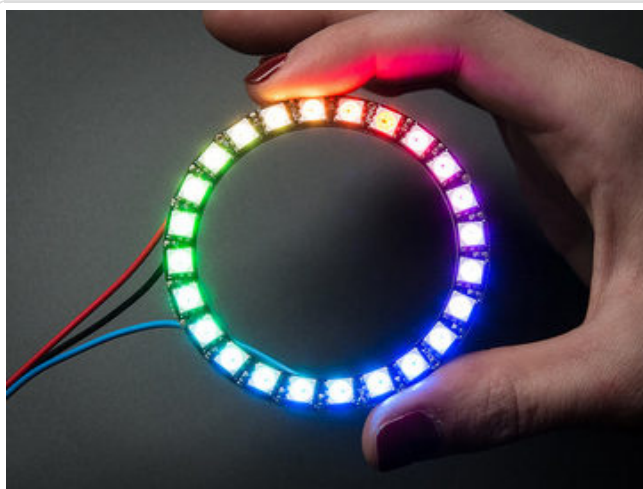
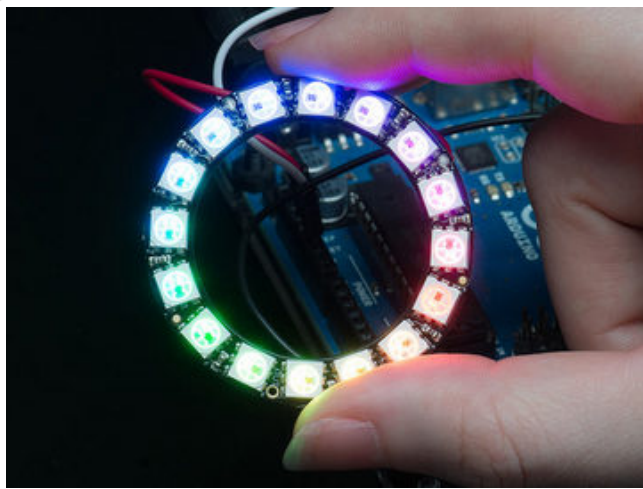
- For 60 and 30 pixels/meter strips, if purchasing less than a full reel (4 or 5 meters, respectively), the strip may or may not have a 3-pin JST plug soldered to one end. These plugs are for factory testing and might be at either end — the plug does not always indicate the input end! Arrows printed on the strip show the actual data direction.
  - Although these strips are flexible, they do not tolerate continuous and repeated bending. A typical application is architectural installations, where they can be curved around columns and then stay put. Repeated flexing (as on costumes) will soon crack the solder connections. For wearable use, either affix shorter segments to a semi-rigid base (e.g. a hat, BMX armor, etc.), or use the individual *sewable* NeoPixels shown later.
  - The flex strips are enclosed in a weatherproof silicone sleeve, making them immune to rain and splashes, but are not recommended for continuous submersion. Early 144 pixel/meter strips were not weatherproof, but the latest batches now include this feature.
  - The silicone sleeve can be cut and removed for a slimmer profile, but the strip is no longer weatherproof.
  - Very few glues will adhere to the weatherproof silicone sleeve. Using zip ties for a “mechanical” bond is usually faster and easier. The only two reliable glues we’ve found are Permatex 66B Clear RTV Silicone (not all silicone glues will work!) and Loctite Plastics Bonding System, a 2-part cyanoacrylate glue. The Permatex 66B silicone can also be used to seal the open end of a cut strip.
  - All strips are manufactured in 1/2 meter segments that are then joined into a longer strip. The pixel spacing across these joins is 2-3 millimeters different than the rest.
-



## NeoPixel Shapes: Rings, Matrix, Shield and Strip

These ready-made shapes simplify many projects! Rings are perfect for jewelry, timepieces and navigation displays. Matrices work for bitmap displays and scrolling message signs.

All NeoPixel shapes can be chained (the output of one can connect to the input of the next) and are controlled from a single microcontroller pin.



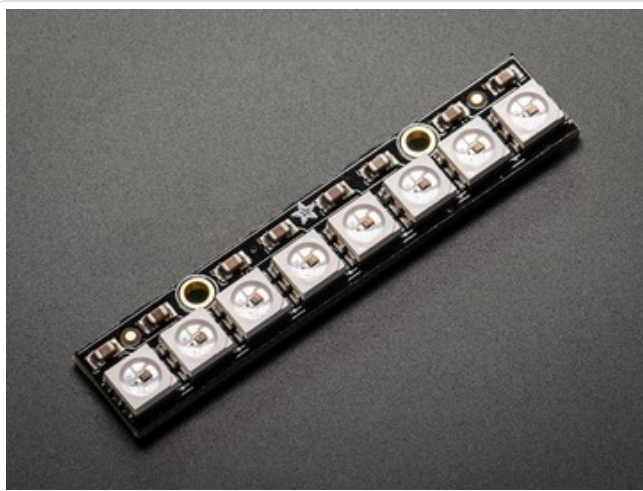
- **12 NeoPixel Ring** (<http://adafru.it/1643>) — 1.5 inch (37 mm) outside diameter. Max 5V @ 750 mA.
- **16 NeoPixel Ring** (<http://adafru.it/1463>) — 1.75 inch (44.5 mm) outside diameter. Max 5V @ 1A.
- **24 NeoPixel Ring** (<http://adafru.it/1586>) — 2.6 inch (66 mm) outside diameter. Max 5V @ 1.4A
- **8x8 NeoMatrix** (<http://adafru.it/1487>) — 64 LEDs total. The pixel spacing and board size allow seamless tiling. Max 5V @ 4A.
- **NeoPixel Shield for Arduino** (<http://adafru.it/1430>) — 5x8 matrix plugs directly atop an Arduino board (can also be tiled separately if desired). If the overall brightness is carefully limited, can be directly powered from the Arduino. For most situations, connecting an external power supply is recommended. Max 5V @ 2.4A (the Arduino can only supply about 500 milliamps).
- **NeoPixel Stick** (<http://adafru.it/1426>) — 8 NeoPixel LEDs on a rigid circuit board packed even more tightly than the flex strips. Solder pads on the back for





connecting wires or breadboard pins (not included).

Current ratings shown are *approximate maximums* (all pixels on at peak brightness). Mixed colors and lower brightness settings will use proportionally less power.



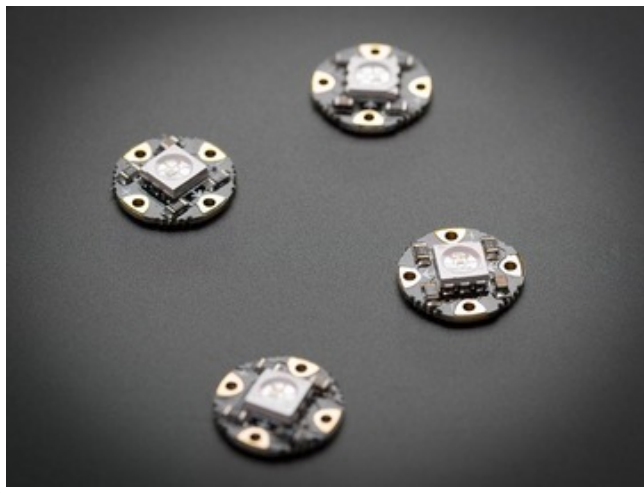
---

## Important Things to Know About NeoPixel Rings

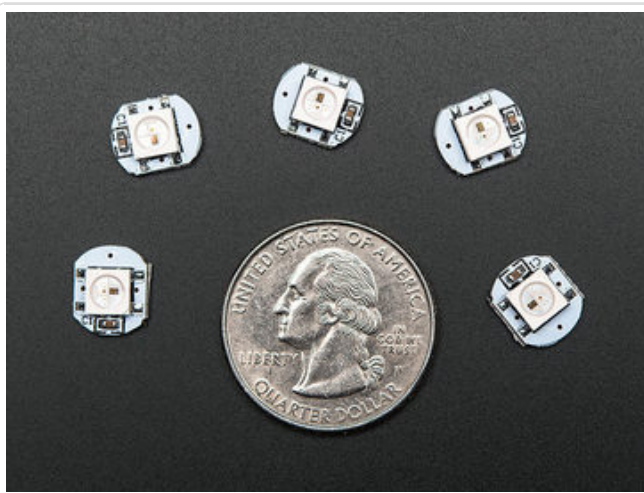
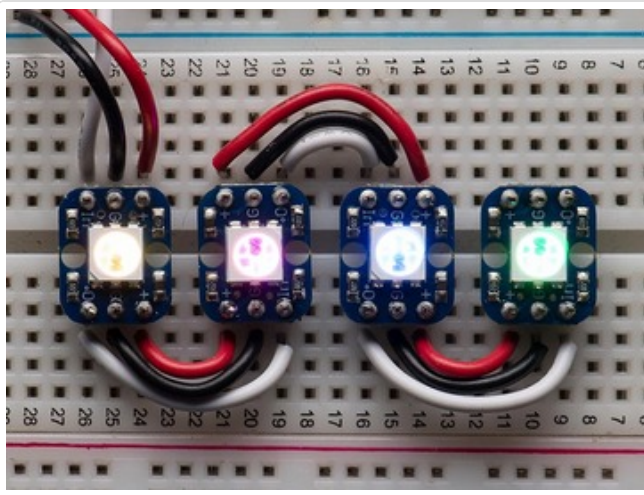
- When soldering wires to these rings, you need to be extra vigilant about solder blobs and short circuits. The spacing between components is *very tight*!
- If using alligator clips, we recommend first soldering short jumper wires to the ring inputs and connecting the clips to those, for similar reasons. *(Some of our tutorials and product photos do show alligator clips directly connected to the rings, but*

*we've had a lot of experience using them.)*

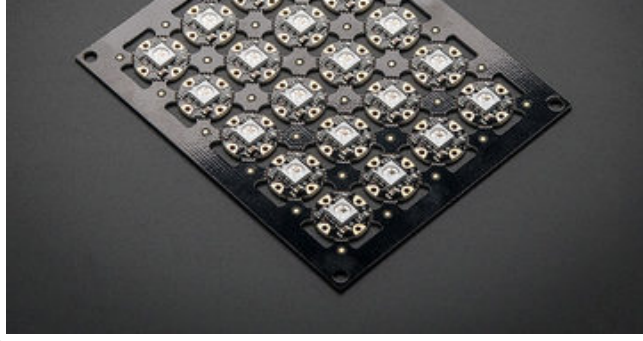
---



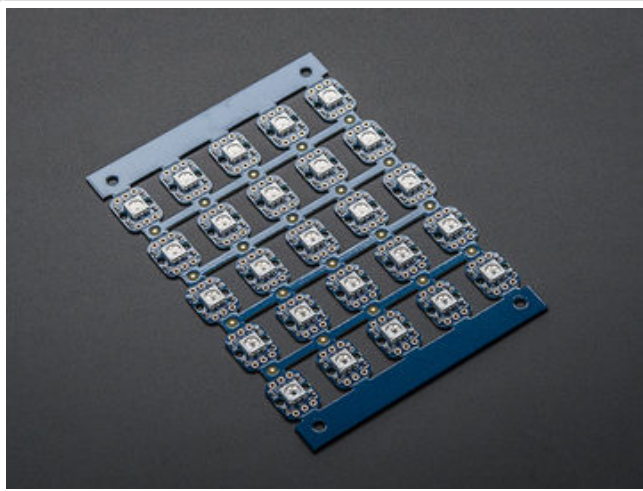
**Single NeoPixels** allow maximum control over individual pixel placement (some are even sewable!), while **bulk sheets** are a more economical option.



- **Flora RGB Smart NeoPixel (Version 2)** (<http://adafruit.it/1260>) — designed specifically for wearables and “soft circuits,” these NeoPixels can be sewn using our [stainless conductive thread](http://adafruit.it/641) (<http://adafruit.it/641>) (traditional soldering works too). Sold in packs of 4.
- **Breadboard-Friendly Smart NeoPixel** (<http://adafruit.it/1312>) — accommodates pin headers with standard 0.1" spacing. Sold in packs of 4 (headers are sold separately).
- **NeoPixel Mini PCB** (<http://adafruit.it/1612>) — the smallest NeoPixel breakout boards have solder pads on the back. Sold in packs of 5.
- **Diffused 8mm Through-Hole NeoPixel** (<http://adafruit.it/1734>) — provides NeoPixels in a more traditional LED form-factor.
- **Flora RGB Smart NeoPixel (Version 2) — Sheet of 20** (<http://adafruit.it/1559>).
- **Breadboard-Friendly Smart NeoPixel — Sheet of 25** (<http://adafruit.it/1558>).
- **WS2812 RGB LED with Integrated Driver Chip** (<http://adafruit.it/1379>) — bare NeoPixels for advanced users wanting to design their own custom boards and shapes. 10 pack. This is a surface-mount device, not recommended for novice soldering. For through-hole use, consider [Breadboard-Friendly NeoPixels](http://adafruit.it/1312) (<http://adafruit.it/1312>).
- **WS2811 LED Driver Chip** (<http://adafruit.it/1378>) — contains the NeoPixel controller circuit without the actual LEDs (can be added separately). 10 pack.



- **Flora RGB Smart NeoPixel (Version 1) — Discontinued**, but listed here for compatibility info. This earlier version of our Flora NeoPixel is identified by a small driver chip on the back side. V1 pixels were originally programmed using the Adafruit\_FloraPixel library. All new code should instead use the [Adafruit\\_NeoPixel library](http://adafru.it/aZU) (<http://adafru.it/aZU>), which handles either type.

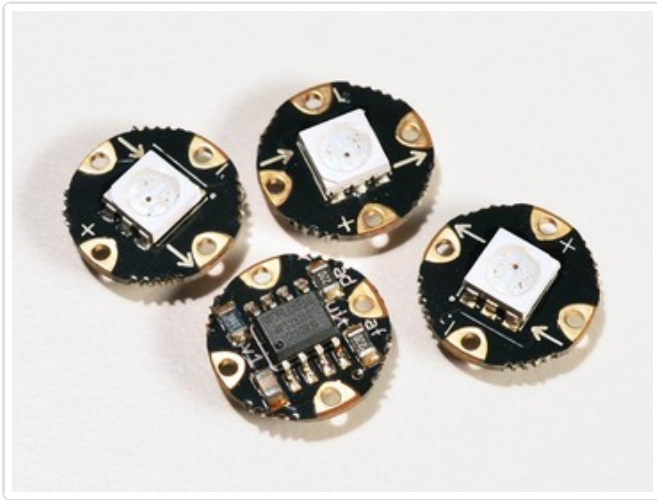


All individual NeoPixels can be chained from a single microcontroller pin, with the one caveat that WS2812 and WS2811 (e.g. Flora V2 and V1) types can't be mixed in a single chain.

Each individual NeoPixel requires 60 milliamps max at full brightness. Mixed colors and lower brightness settings will use proportionally less power.



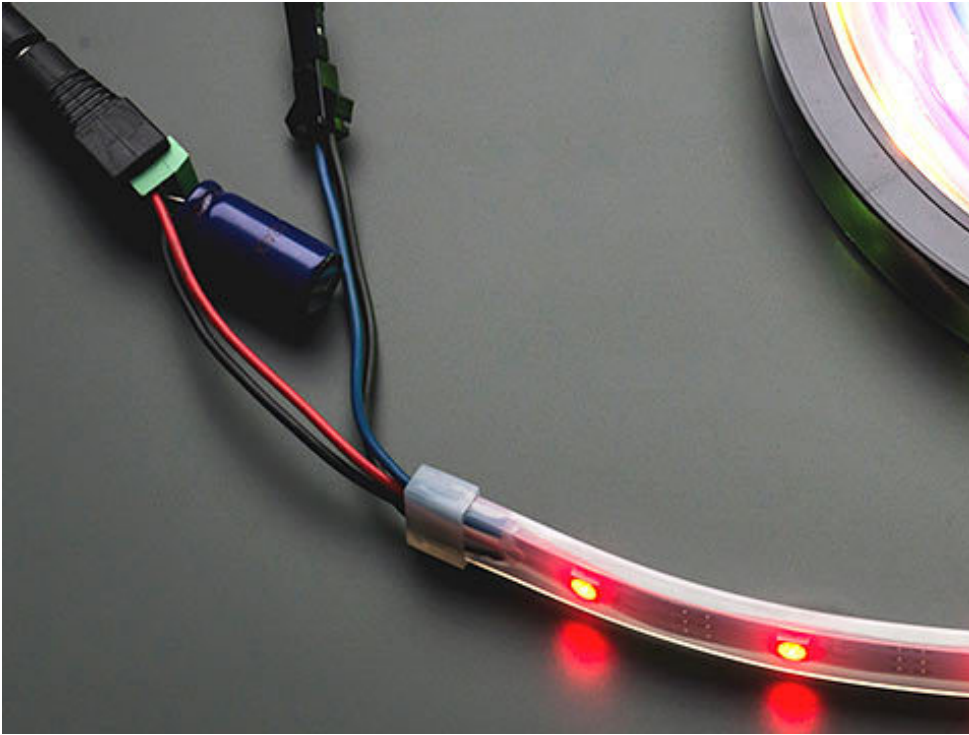




## Best Practices

---

**Improper use can damage your NeoPixels. Before diving in, be aware of the following:**



- Before connecting NeoPixels to any power source, add a large **capacitor** (1000  $\mu$ F, 6.3V or higher) across the + and – terminals as shown above.
- Place a **300 to 500 Ohm resistor** between the Arduino data output pin and the input to the first NeoPixel. This resistor must be at the NeoPixel end of the wire to be effective! *Some products already incorporate this resistor...if you're not sure, add one...there's no harm in doubling up!*
- Try to **minimize the distance** between the Arduino and first pixel.
- **Avoid connecting NeoPixels to a live circuit.** If you simply *must*, always **connect ground first**, then +5V, then data. Disconnect in the reverse order.
- If powering the pixels with a separate supply, apply power to the pixels before applying power to the microcontroller.
- Observe the same precautions as you would for any **static-sensitive** part; ground yourself before handling, etc.
- NeoPixels powered by 5v **require** a 5V data signal. If using a 3.3V microcontroller you must use a logic level shifter such as a 74HCT245. (If you are powering your NeoPixels with 3.7v, a 3.3v data signal is OK)
- Make sure that your connections are secure. Alligator clips do not make reliable connections to the tiny solder pads on NeoPixel rings. Better to solder a small pigtail wire to the ring and attach the alligator clips to that.

Some of our NeoPixel project guides fail to mention the above precautions...they were written before these lessons were learned, and will be updated as required. The design changes are usually minimal (e.g. add inline resistor on data pin in circuit diagrams).

Smaller, battery-operated projects (e.g. FLORA and sewables) are usually fine omitting the capacitor and/or the resistor, but more substantive projects incorporating NeoPixel rings, matrices or a meter or more of NeoPixel strip, or using a plug-in power supply should *definitely* include both!

# Arduino Library

---

Controlling NeoPixels “from scratch” is quite a challenge, so we provide a library letting you focus on the fun and interesting bits. The library works with most mainstream Arduino boards and derivatives: Uno, Mega, Leonardo, Micro, Adafruit Flora, etc. — most anything with an Atmel AVR 8-bit processor from 8 to 16 MHz — and also works with the Arduino Due and all varieties of the PJRC Teensy boards.

Because processor-specific assembly language is used, this library does not work on Netduino, ChipKIT or other advanced “Arduino-like” boards. Others may have written code and libraries for such boards, but we can’t provide technical support for any bugs or trouble there; that’s frontier stuff. Some of this is covered in the “Advanced Coding” section.

Installation of the library is as follows:

1. Visit the [Adafruit\\_NeoPixel library page \(http://adafru.it/aZU\)](http://adafru.it/aZU) at Github.com.
2. Select the “Download ZIP” button, or simply [click this link \(http://adafru.it/cDj\)](http://adafru.it/cDj) to download directly.
3. Uncompress the ZIP file after it’s finished downloading.
4. The resulting folder should contain the files “Adafruit\_NeoPixel.cpp”, “Adafruit\_NeoPixel.h” and an “examples” sub-folder. Sometimes in Windows you’ll get an intermediate-level folder and need to move things around.
5. Rename the folder (containing the .cpp and .h files) to “Adafruit\_NeoPixel” (with the underscore and everything), and place it alongside your other Arduino libraries, typically in your (home folder)/Documents/Arduino/Libraries folder. Libraries should not be installed alongside the Arduino application itself.
6. Re-start the Arduino IDE if it’s currently running.

[Here’s a tutorial \(http://adafru.it/aYM\)](http://adafru.it/aYM) that walks through the process of correctly installing Arduino libraries.

## Basic Connections

---

To get started, let’s assume you have some model of Arduino microcontroller connected to the computer’s USB port. We’ll elaborate on the finer points of powering NeoPixels later, but for now you should use a separate 5V DC power supply (or a 3.7V lithium-ion battery for a Flora wearable project).

Identify the “input” end of your NeoPixel strip, pixel(s) or other device. On some, there will be a solder pad labeled “DIN” or “DI” (data input). Others will have an arrow showing the direction that data moves. The data input can originate from any digital pin on the Arduino, but all the example code is set up for digital pin 6 by default. The NeoPixel shield comes wired this way.

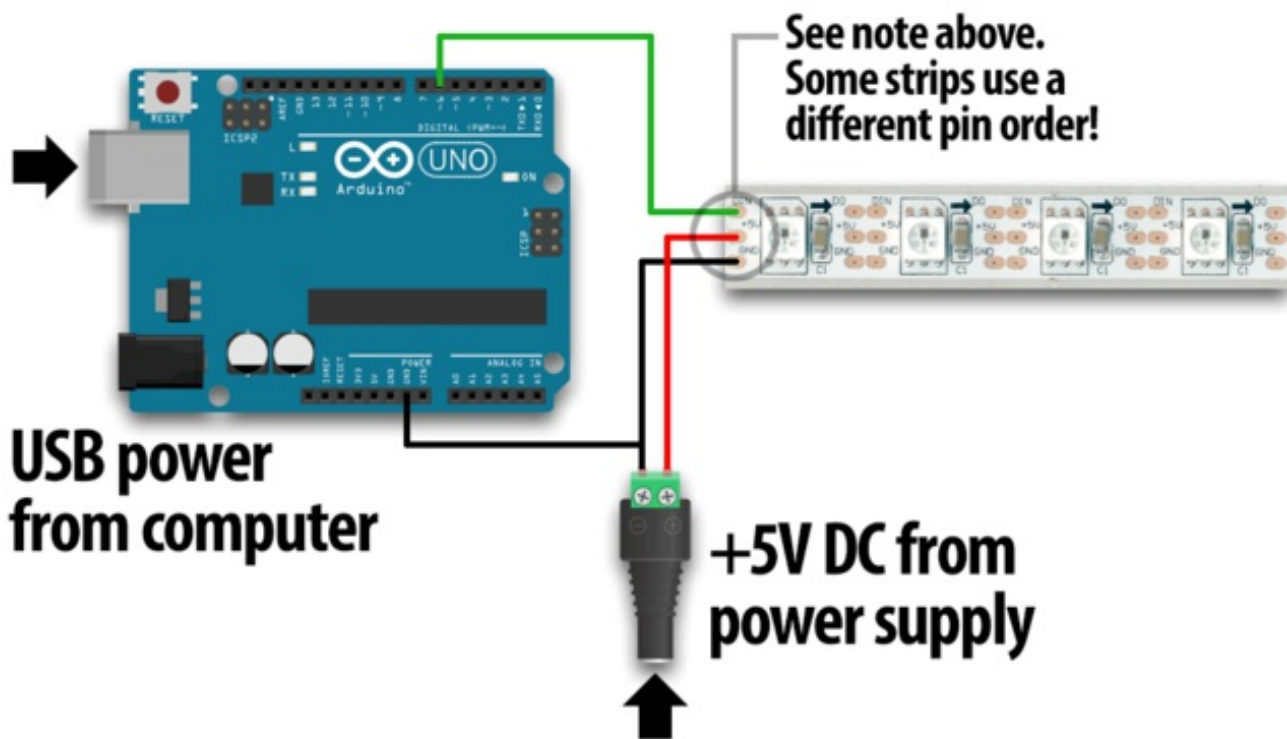
**If using Flora with an attached lithium-ion battery:** connect the +5V input on the strip to the VBATT pad on Flora, GND from the strip to any GND pad on Flora, and DIN to Flora pin D6.



**For other Arduino boards with a separate +5V DC power supply for the NeoPixels:** connect the +5V input on the strip to the + (positive) terminal on the power supply (don't connect to the Arduino), DIN to digital pin 6 on the Arduino, and – (minus or GND) on the strip must connect to both the minus (–) terminal on the DC supply and a GND pin on the Arduino (there are usually several — any will do).

The 144 pixel strips are so tightly packed, there's no room for labels other than –, + and the data direction arrows. Data is the un-labeled pad.

The order of the three pins can vary between different strip densities and batches. ALWAYS use the labels printed ON THE STRIP. Look closely, NEVER blindly follow a NeoPixel strip wiring diagram; it might be based on a different strip type!



When connecting NeoPixels to any live power source or microcontroller, ALWAYS CONNECT GROUND (–) BEFORE ANYTHING ELSE. Conversely, disconnect ground last when separating.

Adding a ~470 ohm resistor between your microcontroller's data pin and the data input on the NeoPixels can help prevent spikes on the data line that can damage your first pixel. Please add one between your micro and NeoPixels!

We also recommend adding a large capacitor (1000  $\mu$ F, 6.3V or higher) across the + and – terminals. This prevents the initial onrush of current from damaging the pixels. See the photo on the next page for an example.

Can NeoPixels be powered directly from the Arduino's 5V pin?

**Sometimes.** The Arduino can continuously supply only about 500 milliamps to the 5V pin. Each NeoPixel can draw up to 60 milliamps at full brightness. So yes, you can skip the separate DC supply and power directly off the Arduino *as long as just a few pixels are used*, or more if the colors and overall brightness are low. When in doubt, give the pixels a separate power supply.

## A Simple Code Example: strandtest

Launch the Arduino IDE. From the **File** menu, select

**Sketchbook**→**Libraries**→**Adafruit\_NeoPixel**→**strandtest**

(If the Adafruit\_NeoPixel rollover menu is not present, the library has not been correctly installed, or the IDE needs to be restarted after installation. Check the installation steps above to confirm it's properly named and located.)

Select your board type and serial port from the **Tools** menu, and try uploading to the board. If the NeoPixels are connected and powered, you should see a little light show.

Nothing happens!

Check your connections. The most common mistake is connecting to the output end of a strip rather than the input.

Let's look at the code now...

All NeoPixel sketches begin by including the header file:

```
#include <Adafruit_NeoPixel.h>
```

The block of code that follows is mostly descriptive comments. Only the last line is really doing any work:

```
#define PIN 6

// Parameter 1 = number of pixels in strip
// Parameter 2 = pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_KHZ800 800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400 400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
```

```
// NEO_GRB   Pixels are wired for GRB bitstream (most NeoPixel products)
// NEO_RGB   Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
Adafruit_NeoPixel strip = Adafruit_NeoPixel(60, PIN, NEO_GRB + NEO_KHZ800);
```

The first line assigns a number to the symbol “PIN” for later reference. It doesn’t *need* to be done this way, but makes it easier to change the pin where the NeoPixels are connected without digging deeper into the code.

The last line declares a NeoPixel *object*. We’ll refer to this by name later to control the strip of pixels. There are three parameters or *arguments* in parenthesis:

1. The number of sequential NeoPixels in the strip. In the example this is set to 60, equal to 1 meter of medium-density strip. Change this to match the actual number you’re using.
2. The pin number to which the NeoPixel strip (or other device) is connected. Normally this would be a number, but we previously declared the symbol PIN to refer to it by name here.
3. A value indicating the type of NeoPixels that are connected. **In most cases you can leave this off and pass just two arguments;** the example code is just being extra descriptive. If you have a supply of classic “V1” Flora pixels, those require NEO\_KHZ400 + NEO\_RGB to be passed here.

For through-hole 8mm NeoPixels, use NEO\_RGB instead of NEO\_GRB in the strip declaration.

Then, in the setup() function, call begin() to prepare the data pin for NeoPixel output:

```
void setup() {
  strip.begin();
  strip.show(); // Initialize all pixels to 'off'
}
```

The second line, strip.show(), isn’t absolutely necessary, it’s just there to be thorough. That function pushes data out to the pixels...since no colors have been set yet, this initializes all the NeoPixels to an initial “off” state in case some were left lit by a prior program.

In the strandtest example, loop() doesn’t set any pixel colors on its own — it calls other functions that create animated effects. So let’s ignore it for now and look ahead, inside the individual functions, to see how the strip is controlled.

There are two ways to set the color of a pixel. The first is:

```
strip.setPixelColor(n, red, green, blue);
```

The first argument — *n* in this example — is the pixel number along the strip, starting from 0 closest to the Arduino. If you have a strip of 30 pixels, they're numbered 0 through 29. It's a computer thing. You'll see various places in the code using a for loop, passing the loop counter variable as the pixel number to this function, to set the values of multiple pixels.

The next three arguments are the pixel color, expressed as red, green and blue brightness levels, where 0 is dimmest (off) and 255 is maximum brightness.

To set the 12th pixel (#11, counting from 0) to magenta (red + blue), you could write:

```
strip.setPixelColor(11, 255, 0, 255);
```

An alternate syntax has just two arguments:

```
strip.setPixelColor(n, color);
```

Here, *color* is a 32-bit type that merges the red, green and blue values into a single number. This is sometimes easier or faster for some (but not all) programs to work with; you'll see the strandtest code uses both syntaxes in different places.

You can also convert separate red, green and blue values into a single 32-bit type for later use:

```
uint32_t magenta = strip.Color(255, 0, 255);
```

Then later you can just pass "magenta" as an argument to `setPixelColor` rather than the separate red, green and blue numbers every time.

`setPixelColor()` does not have an immediate effect on the LEDs. To "push" the color data to the strip, call `show()`:

```
strip.show();
```

This updates the whole strip at once, and despite the extra step is actually a good thing. If every call to `setPixelColor()` had an immediate effect, animation would appear jumpy rather than buttery smooth.

You can query the color of a previously-set pixel using `getPixelColor()`:



```
uint32_t color = strip.getPixelColor(11);
```

This returns a 32-bit merged color value.

The number of pixels in a previously-declared strip can be queried using `numPixels()`:

```
uint16_t n = strip.numPixels();
```

The overall brightness of all the LEDs can be adjusted using `setBrightness()`. This takes a single argument, a number in the range 0 (off) to 255 (max brightness). For example, to set a strip to 1/4 brightness:

```
strip.setBrightness(64);
```

Just like `setPixel()`, **this does not have an immediate effect**. You need to follow this with a call to `show()`.

You can't move from a lower brightness to a higher setting without some loss in fidelity. Certain animation effects are better served by leaving the brightness at max and calling `setPixel()` repeatedly to fill the strip.

I'm calling `setPixel()` but nothing's happening!

There are two main culprits for this:

1. forgetting to call `strip.begin()` in `setup()`.
2. forgetting to call `strip.show()` after setting pixel colors.

Another (less common) possibility is running out of RAM — see the last section below. If the program *sort of* works but has unpredictable results, consider that.

Can I have multiple NeoPixel objects on different pins?

Certainly! Each requires its own declaration with a unique name:

```
Adafruit_NeoPixel strip_a = Adafruit_NeoPixel(16, 5);  
Adafruit_NeoPixel strip_b = Adafruit_NeoPixel(16, 6);
```

The above declares two distinct NeoPixel objects, one each on pins 5 and 6, each containing 16 pixels and using the implied default type (NEO\_KHZ800 + NEO\_GRB).

Can I connect multiple NeoPixel strips to the same Arduino pin?

In many cases, yes. All the strips will then show exactly the same thing. This only works up to a point though...four strips on a single pin is a good and reliable number. If you need more than that, individual NeoPixels can be used as buffers to “fan out” to more strips: connect one Arduino pin to the inputs of four separate NeoPixels, then connect each

pixels' output to the inputs of four strips (or fewer, if you don't need quite that many). If the strips are 10 pixels long, declare the NeoPixel object as having 11 pixels. The extra "buffer" pixels will be at position #0 — just leave them turned off — and the strips then run from positions 1 through 10.

I'm getting the wrong colors. Red and blue are swapped!

When using through-hole 8mm NeoPixels (or V1 Flora pixels), use `NEO_RGB` for the third parameter in the `Adafruit_NeoPixel` declaration. For all other types of NeoPixels, use `NEO_GRB`.

## Pixels Gobble RAM

---

Each NeoPixel requires about 3 bytes of RAM. This doesn't sound like very much, but when you start using dozens or even hundreds of pixels, and consider that the mainstream Arduino Uno only has 2 kilobytes of RAM (often much less after other libraries stake their claim), this can be a real problem!

For using really large numbers of LEDs, you might need to step up to a more potent board like the Arduino Mega or Due. But if you're close and need just a little extra space, you can sometimes tweak your code to be more RAM-efficient. [This tutorial has some pointers on memory usage. \(http://adafru.it/coj\)](http://adafru.it/coj)

# Powering NeoPixels

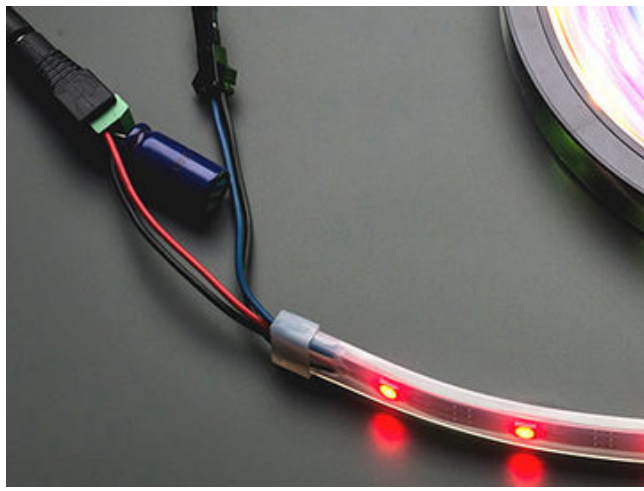
When connecting NeoPixels to any live power source or microcontroller, **ALWAYS CONNECT GROUND (-) BEFORE ANYTHING ELSE**. Conversely, disconnect ground last when separating.

Adding a 300 to 500 Ohm resistor between your microcontroller's data pin and the data input on the first NeoPixel can help prevent voltage spikes that might otherwise damage your first pixel. Please add one between your micro and NeoPixels!

NeoPixels are usually described as “5 Volt devices,” but the reality is a little more nuanced than that.

Some (not all) NeoPixel products can work with slightly higher voltages. This depends on the additional support components around the chip, based on available space, cost and the most likely application. **Refer to the specific product description page for guidance on acceptable voltage limits for each type.** When in doubt, aim for 5 Volts.

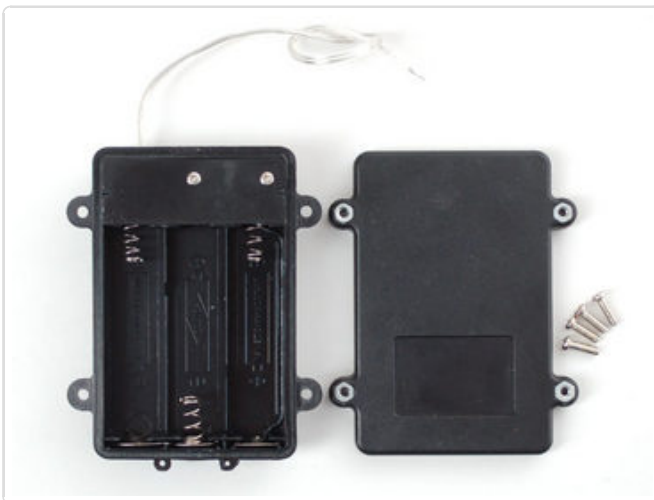
**Lower voltages are always acceptable**, with the caveat that the LEDs will be slightly dimmer. There's a limit below which the LED will fail to light, or will start to show the wrong color.



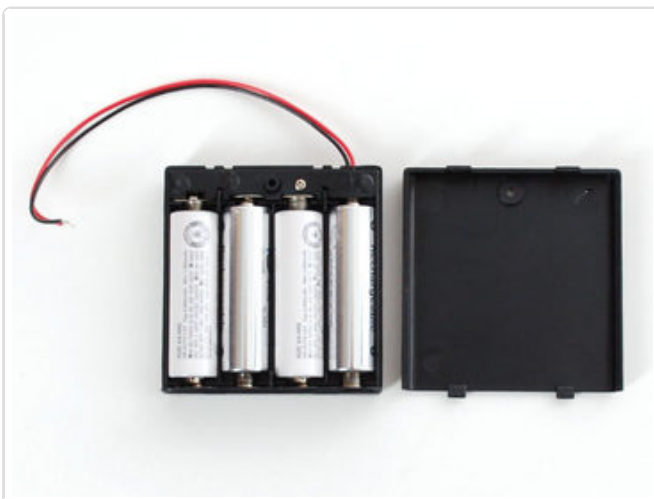
**Before connecting a NeoPixel strip to ANY source of power, we very strongly recommend adding a large capacitor (1000  $\mu$ F, 6.3V or higher) across the + and - terminals. This prevents the initial onrush of current from damaging the pixels.**



For many wearable projects we recommend a [lithium-polymer battery](http://adafruit.it/328) (<http://adafruit.it/328>). These deliver 3.7 Volts — perfect for directly feeding low-power microcontrollers such as the Adafruit Flora, yet enough voltage to run a short length of NeoPixels.



Three alkaline cells (such as AA batteries) can be installed in a [battery holder](http://adafruit.it/771) (<http://adafruit.it/771>) to provide 4.5 Volts. Though larger and heavier than the fancy lithium-polymer pack, they're inexpensive and readily available.



Four nickel-metal hydride (NiMH) rechargeable cells can similarly be used in a [4-cell battery holder](http://adafruit.it/830) (<http://adafruit.it/830>) to provide 4.8 Volts.

**Make sure you only use NiMH cells in this configuration.** Four alkaline cells (the disposable type) will output 6V total — that's too high for some NeoPixels, and definitely too much for the microcontroller!

**Battery-operated LED project planning is discussed in greater detail in**





For most non-portable “desktop” projects, a [5V DC switching power supply](http://adafru.it/276) (<http://adafru.it/276>) is ideal. This small 2 Amp supply is good for a meter or so of NeoPixel strip. We’ll explain larger projects in a moment.



**Be extremely cautious with bench power supplies.** Some — even reputable, well-regarded brands — can produce a large voltage spike when initially switched on, instantly destroying your NeoPixels!

If you use a bench supply, do not connect NeoPixels directly. Turn on the power supply first, let the voltage stabilize, *then* connect the pixels (GND first).

---

## Estimating Power Requirements

Each individual NeoPixel draws up to 60 milliamps at maximum brightness white (red + green + blue). In actual use though, it’s rare for all pixels to be turned on that way. When mixing colors and displaying animations, the current draw will be much less. It’s impossible to estimate a single number for all circumstances, but we’ve been using 1/3 this (20 mA per pixel) as a gross rule of thumb with no ill effects. But if you know for a fact that you need every pixel on at maximum brightness, use the full 60 mA figure.

To estimate power supply needs, multiply the number of pixels by 20, then divide the result by 1,000 for the “rule of thumb” power supply rating in Amps. Or use 60 (instead of 20) if you want to guarantee an absolute margin of safety for all situations. For example:

$60 \text{ NeoPixels} \times 20 \text{ mA} \div 1,000 = 1.2 \text{ Amps minimum}$

$60 \text{ NeoPixels} \times 60 \text{ mA} \div 1,000 = 3.6 \text{ Amps minimum}$

The choice of “overhead” in your power supply is up to you. Maximum safety and reliability are achieved with a more generously-sized power supply, and this is what we recommend. Most power supplies can briefly push a little extra current for short periods. Many contain a thermal fuse and will simply shut down if overworked. So they may technically *work*, but this is the electronics equivalent of abusing a rental car.

I estimate I need a 3.6 Amp power supply. I have a 10 Amp supply on-hand. Will this cause my NeoPixels to explode?

As long as the output is 5 Volts DC, you’re golden. The LEDs will only draw as much current (Amperes) as they need. So extra Amps are OK — in fact, it can be a *good* thing. The larger power supply will run cooler because it’s not being pushed to its limit.

Excessive *voltage*, however, will definitely kill your LEDs.

**Extra Amps = good. Extra Volts = bad.**

What about batteries and “Amp hours”?

Amp-hours are current over time. A 2,600 mAh (milliamp-hour) battery can be thought of as delivering 2.6 Amps continuously for one hour, or 1.3 Amps for 2 hours, and so forth. In reality, it’s not quite linear like that; most batteries have disproportionately shorter run times with a heavy load. Also, most batteries won’t take kindly to being discharged in an hour — **this can even be dangerous!** Select a battery sufficiently large that it will take at least a couple hours to run down. It’s both safer for you and better for the longevity of the battery.

I need to power LOTS of NeoPixels and don’t have a power supply that large. Can I use several smaller ones?

**Maybe.** There are benefits to using a single supply, and large power supplies are discussed below. “Non-optimal” doesn’t necessarily mean “pessimal” though, and we wouldn’t discourage anyone from using what resources they have.

If you go this route, the key is to have all of the ground pins among the strips connected in common, but the +5V from each power supply should be connected only to one length of NeoPixels — those should *not* all be joined. Every power supply is a little different — not *precisely* 5 Volts — and this keeps some from back-feeding into others.

## Giant Power Supplies

Adafruit offers 5V DC power supplies up to **10 Amps** (<http://adafru.it/658>). This is usually sufficient for a couple hundred NeoPixels or more. For *really* large installations, you’ll need to look elsewhere.

One possibility is to repurpose an ATX computer power supply. The nice beefy server types often provide up to 30 Amps. Some minor modifications are needed...Google around for

“ATX power supply hack.”

Even larger (and scarier, and much more expensive) are laboratory power supplies with ratings into the *hundreds* of Amps. Sometimes this is what’s needed for architectural scale projects and large stage productions. And occasionally we get requests for help...

Please note that **projects of this scale are potentially very dangerous**, and the problems of power distribution are fundamentally different than hobby-scale projects. As much as we enjoy helping our customers in the forums, they are for *product technical support* and not full-on *engineering services*. If you’re developing a project of this scope, hire a professional electrician with experience in high-power, low-voltage systems such as photovoltaics or large RVs and boats. This is no charade.

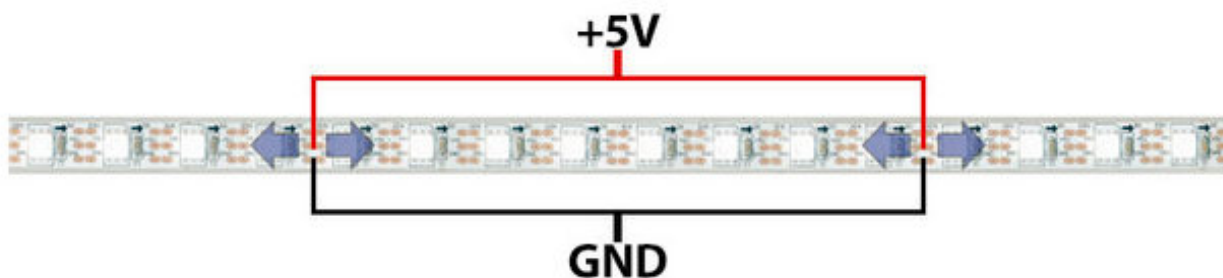
## Distributing Power

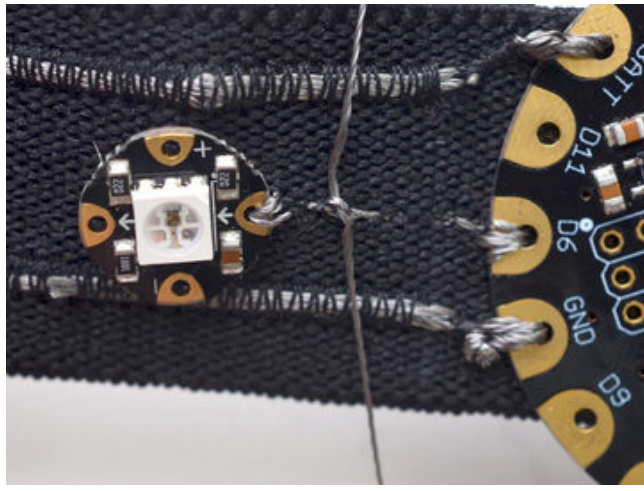
The longer a wire is, the more resistance it has. The more resistance, the more voltage drops along its length. If voltage drops too far, the color of NeoPixels can be affected.

Consider a full 4 meter reel of NeoPixels. With 5V applied at one end of the strip, for those pixels closest to this end, power traverses only a few inches of copper. But at the far end of the strip, power traverses *8 meters* of copper — 4 meters out on the +5V line, 4 meters back on the ground line. Those furthest pixels will be tinted brown due to the voltage drop (blue and green LEDs require higher voltage than red).



**Pro Tip:** NeoPixels don’t care what end they receive power from. Though data moves in only one direction, electricity can go either way. You can connect power at the head, the tail, in the middle, or ideally distribute it to several points. Try to aim for about 1 meter lengths for the best color consistency. With larger NeoPixel setups, think of power distribution as branches of a tree rather than one continuous line.





Resistance is just as much a concern on tiny projects too!

For wearable electronics we like conductive thread...it's flexible and withstands hand washing. Downside is that it doesn't carry much current. Here several strands of conductive thread have been grouped to provide better capacity for the + and - conductors down a pair of suspenders.

(From the [Pac Man Pixel Suspenders](http://adafru.it/ciD) (<http://adafru.it/ciD>) guide.)

---

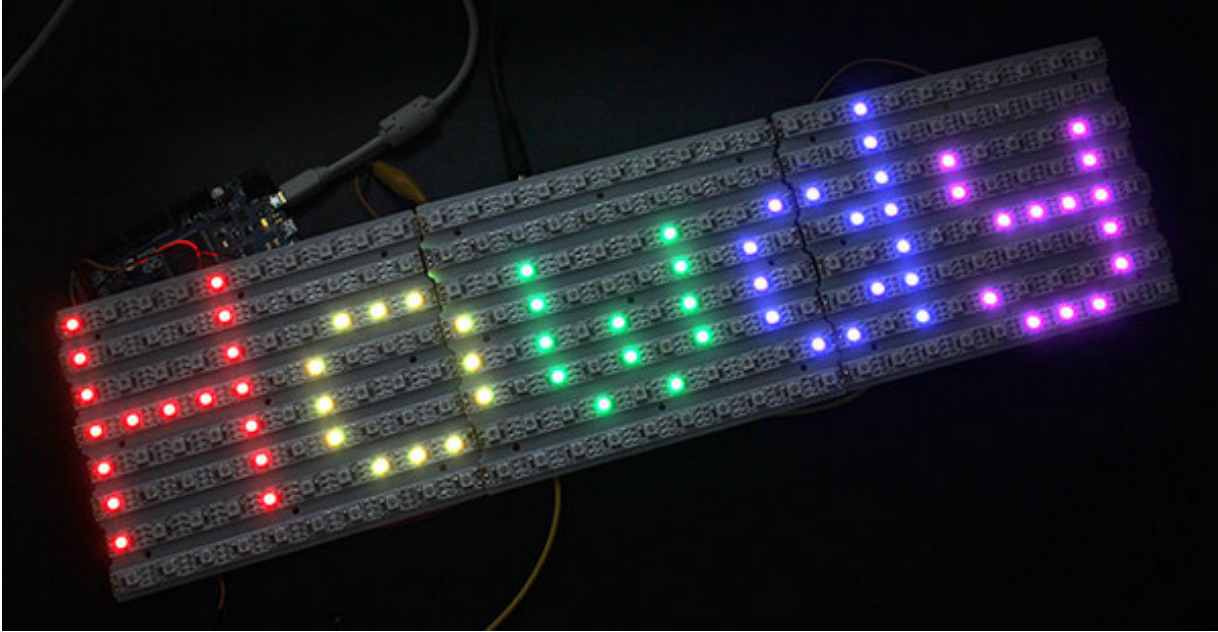
## Driving 5V NeoPixels from 3.3V Microcontrollers

Increasingly, microcontrollers are running at 3.3 Volts instead of 5 Volts. That's great news for efficiency, but can present a communication problem with 5V NeoPixels. The 3.3V signal from the microcontroller may not be "loud" enough to register with the higher-voltage device. The manufacturer recommends a minimum signal voltage of 70% of the NeoPixel voltage.

There are two ways this can be addressed:

1. Lower the voltage to the NeoPixels so it's closer (or equal) to that of the microcontroller. This is why we recommend LiPo batteries for FLORA projects: 3.7V is enough to run a short length of pixels, and the microcontroller is comfortable at that voltage as well.
2. Use a [logic level shifter](http://adafru.it/757) (<http://adafru.it/757>) to step up the signal from the microcontroller to the first pixel.

# NeoMatrix Library



The Adafruit\_NeoMatrix library builds upon Adafruit\_NeoPixel to create two-dimensional graphic displays using NeoPixels. You can then easily draw shapes, text and animation without having to calculate every X/Y pixel position. Small NeoPixel matrices are available in the shop. Larger displays can be formed using sections of NeoPixel strip, as shown in the photo above.

In addition to the Adafruit\_NeoPixel library (which was already downloaded and installed in a prior step), NeoMatrix requires two additional libraries:

1. [Adafruit\\_NeoMatrix](http://adafru.it/cDt) (<http://adafru.it/cDt>)
2. [Adafruit\\_GFX](http://adafru.it/cBB) (<http://adafru.it/cBB>)

If you've previously used any Adafruit LCD or OLED displays, you might already have the latter library installed.

Installation for both is similar to Adafruit\_NeoPixel before: unzip, make sure the folder name matches the .cpp and .h files within, then move to your Arduino libraries folder and restart the IDE.

Arduino sketches need to include all three headers just to use this library:

```
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>
#ifndef PSTR
#define PSTR // Make Arduino Due happy
#endif
```



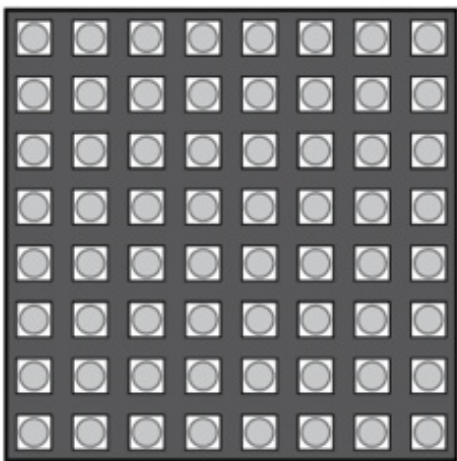
The extra `ifdef/define/endif` lines are only needed if you're using an Arduino Due. Otherwise they can be left out.

## Layouts

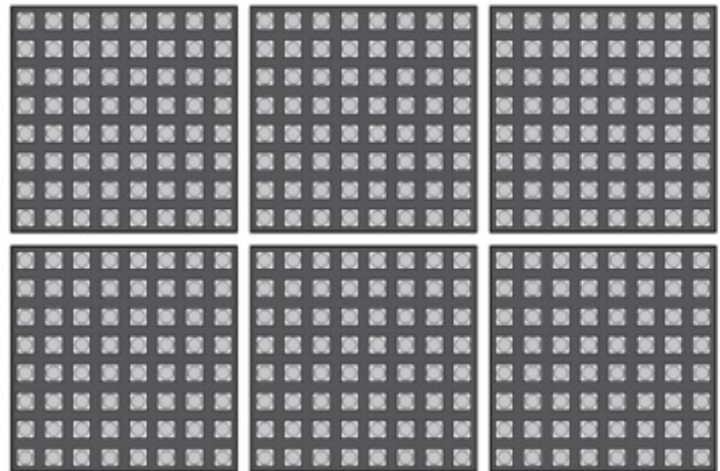
Adafruit\_NeoMatrix uses exactly the same coordinate system, color functions and graphics commands as the Adafruit\_GFX library. If you're new to the latter, [a separate tutorial explains its use](http://adafru.it/aPe) (<http://adafru.it/aPe>). There are also example sketches included with the Adafruit\_NeoMatrix library.

We'll just focus on the *constructor* here — how to declare a two-dimensional display made from NeoPixels. Powering the beast is another matter, covered on the prior page.

The library handles both *single* matrices — all NeoPixels in a single uniform grid — and *tiled* matrices — multiple grids combined into a larger display:

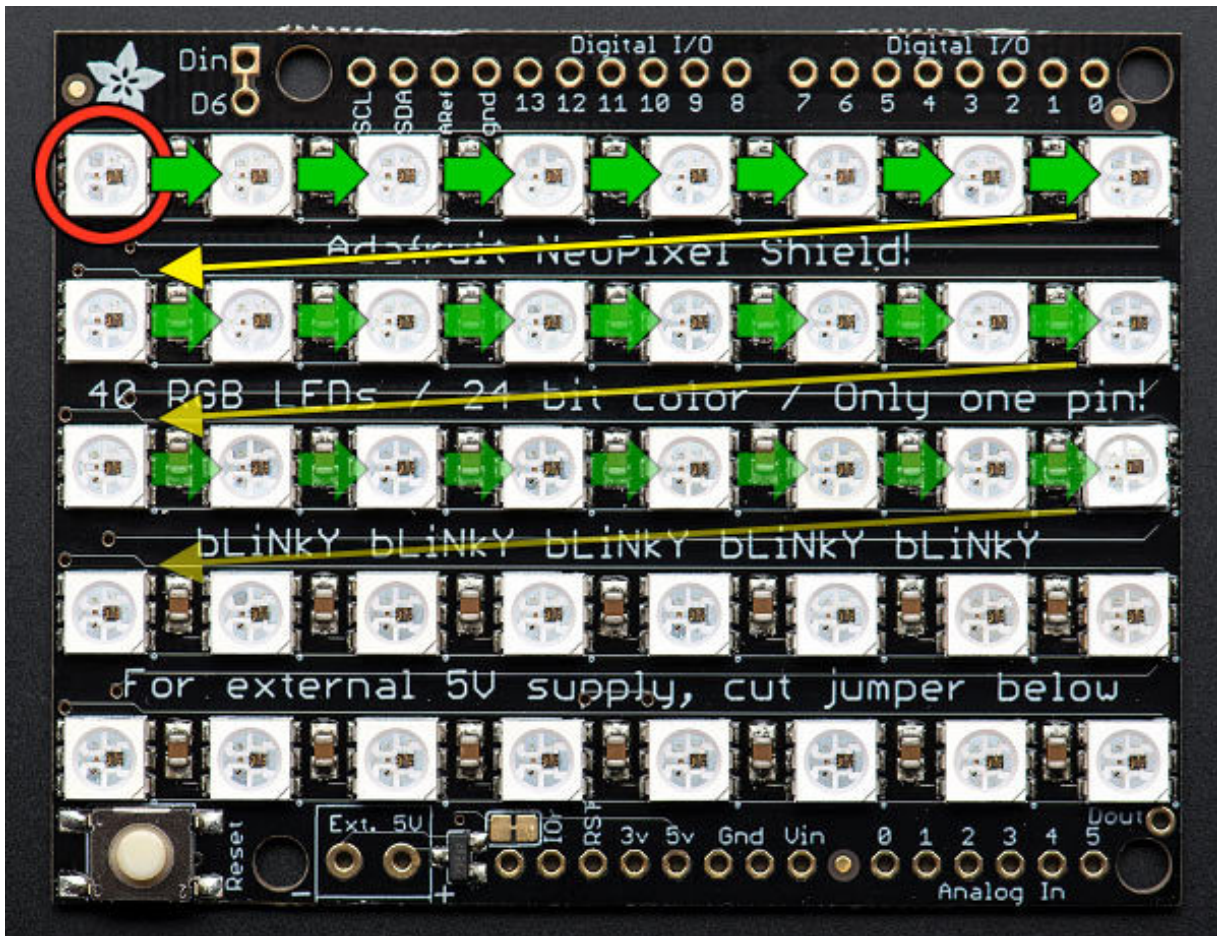


### Single Matrix



### Tiled Matrices

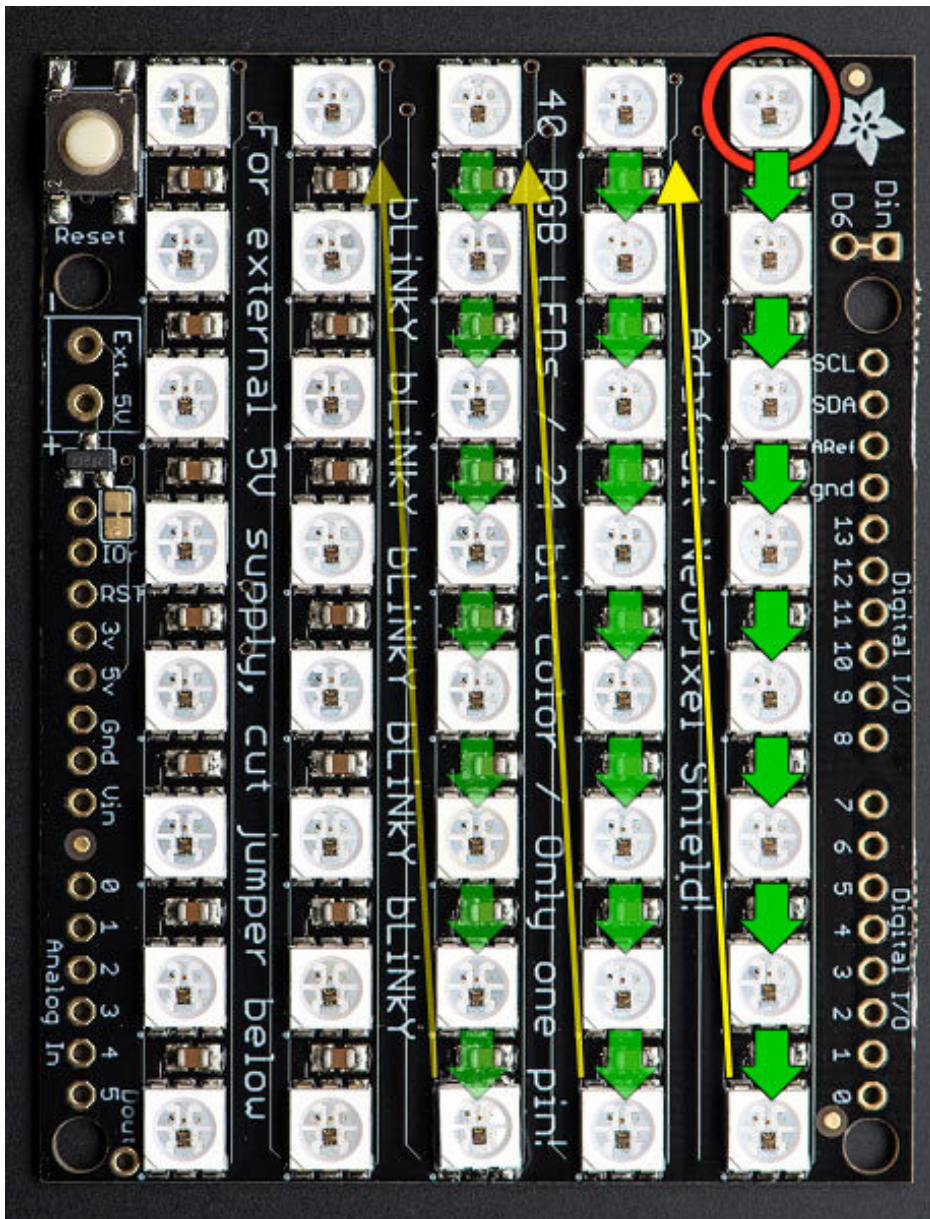
Let's begin with the declaration for a single matrix, because it's simpler to explain. We'll be demonstrating the NeoPixel Shield for Arduino in this case — an 8x5 matrix of NeoPixels. When looking at this shield with the text in a readable orientation, the first pixel, #0, is at the top left. Each successive pixel is right one position — pixel 1 is directly to the right of pixel 0, and so forth. At the end of each row, the next pixel is at the left side of the next row. This isn't something we decide in code...it's how the NeoPixels are hard-wired in the circuit board comprising the shield.



We refer to this layout as *row major* and *progressive*. *Row major* means the pixels are arranged in horizontal lines (the opposite, in vertical lines, is *column major*). *Progressive* means each row proceeds in the same direction. Some matrices will reverse direction on each row, as it can be easier to wire that way. We call that a *zigzag* layout.

However...for this example, we want to use the shield in the “tall” direction, so the Arduino is standing up on the desk with the USB cable at the top. When we turn the board this way, the matrix layout changes...





Now the first pixel is at the **top right**. Pixels increment top-to-bottom — it's now **column major**. The order of the columns is still **progressive** though.

We declare the matrix thusly:

```
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(5, 8, 6,
  NEO_MATRIX_TOP + NEO_MATRIX_RIGHT +
  NEO_MATRIX_COLUMNS + NEO_MATRIX_PROGRESSIVE,
  NEO_GRB + NEO_KHZ800);
```

The first two arguments — 5 and 8 — are the width and height of the matrix, in pixels. The third argument — 6 — is the pin number to which the NeoPixels are connected. On the shield this is hard-wired to digital pin 6, but standalone matrices are free to use other pins.

The next argument is the interesting one. This indicates where the first pixel in the matrix is positioned and the arrangement of rows or columns. The first pixel must be at one of the four corners; *which* corner is indicated by adding either NEO\_MATRIX\_TOP or NEO\_MATRIX\_BOTTOM to either NEO\_MATRIX\_LEFT or NEO\_MATRIX\_RIGHT. The row/column arrangement is indicated by further adding either NEO\_MATRIX\_COLUMNS or NEO\_MATRIX\_ROWS to either NEO\_MATRIX\_PROGRESSIVE or NEO\_MATRIX\_ZIGZAG. These values are all added to form a single value as in the above code.

```
NEO_MATRIX_TOP + NEO_MATRIX_RIGHT + NEO_MATRIX_COLUMNS +  
NEO_MATRIX_PROGRESSIVE
```

The last argument is exactly the same as with the NeoPixel library, indicating the type of LED pixels being used. In the majority of cases with the latest NeoPixel products, you can simply leave this argument off...the example code is just being extra descriptive.

**The point of this setup is that the rest of the sketch never needs to think about the layout of the matrix. Coordinate (0,0) for drawing graphics will always be at the top-left, regardless of the actual position of the first NeoPixel.**

Why not just use the rotation feature in Adafruit\_GFX?

Adafruit\_GFX only handles rotation. Though it would handle our example above, it doesn't cover every permutation of rotation *and mirroring* that may occur with certain matrix layouts, not to mention the zig-zag capability, or this next bit...

## Tiled Matrices

A *tiled* matrix is comprised of multiple smaller NeoPixel matrices. This is sometimes easier for assembly or for distributing power. All of the sub-matrices need to be the same size, and must be ordered in a predictable manner. The Adafruit\_NeoMatrix() constructor then receives some additional arguments:

```
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(  
  matrixWidth, matrixHeight, tilesX, tilesY, pin, matrixType, ledType);
```

The first two arguments are the width and height, in pixels, of each tiled sub-matrix, not the entire display.

The next two arguments are the number of tiles, in the horizontal and vertical direction. The dimensions of the overall display then will always be a multiple of the sub-matrix dimensions.

The fifth argument is the pin number, same as before and as with the NeoPixel library. The last argument also follows prior behaviors, and in most cases can be left off.

The second-to-last argument though...this gets complicated...

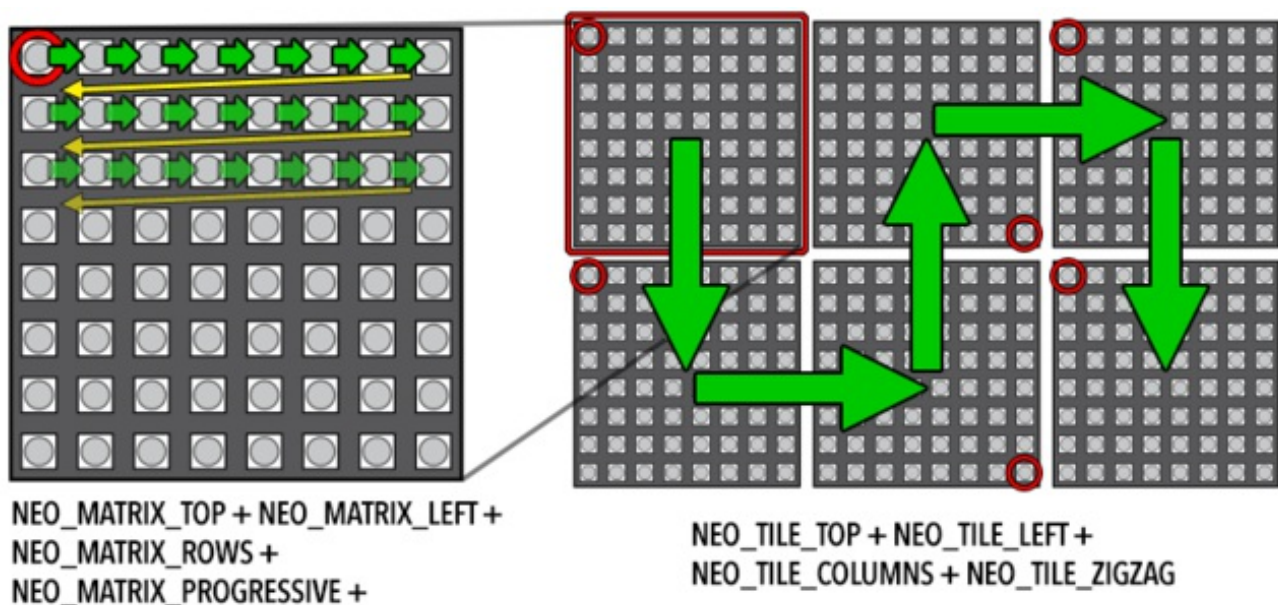
With a single matrix, there was a starting corner, a major axis (rows or columns) and a line sequence (progressive or zigzag). This is now doubled — similar information is needed both for the pixel order within the individual tiles, and the overall arrangement of tiles in the display. As before, we add up a list of symbols to produce a single argument describing the display format.

The NEO\_MATRIX\_\* symbols work the same as in the prior single-matrix case, and now refer to the individual sub-matrices within the overall display. All tiles must follow the same format. An additional set of symbols work similarly to then describe the tile order.

The first tile must be located at one of the four corners. Add either NEO\_TILE\_TOP or NEO\_TILE\_BOTTOM and NEO\_TILE\_LEFT or NEO\_TILE\_RIGHT to indicate the position of the first tile. This is independent of the position of the first pixel within the tiles; they can be different corners.

Tiles can be arranged in horizontal rows or vertical columns. Again this is independent of the pixel order within the tiles. Add either NEO\_TILE\_ROWS or NEO\_TILE\_COLUMNS.

Finally, rows or columns of tiles may be arranged in progressive or zigzag order; that is, every row or column proceeds in the same order, or alternating rows/columns switch direction. Add either NEO\_TILE\_PROGRESSIVE or NEO\_TILE\_ZIGZAG to indicate the order. **BUT**...if NEO\_TILE\_ZIGZAG order is selected, alternate lines of tiles must be rotated 180 degrees. This is intentional and by design; it keeps the tile-to-tile wiring more consistent and simple. This rotation is not required for NEO\_TILE\_PROGRESSIVE.



Tiles don't need to be square! The above is just one possible layout. The display shown at the top of this page is three 10x8 tiles assembled from NeoPixel strip.

Once the matrix is defined, the remainder of the project is similar to Adafruit\_NeoPixel. Remember to use `matrix.begin()` in the `setup()` function and `matrix.show()` to update the display after drawing. The `setBrightness()` function is also available. The library includes a couple of example sketches for reference.

## Other Layouts

For any other cases that are not uniformly tiled, you can provide your own function to remap X/Y coordinates to NeoPixel strip indices. This function should accept two unsigned 16-bit arguments (pixel X, Y coordinates) and return an unsigned 16-bit value (corresponding strip index). The simplest row-major progressive function might resemble this:

```
uint16_t myRemapFn(uint16_t x, uint16_t y) {  
    return WIDTH * y + x;  
}
```

That's a crude example. Yours might be designed for pixels arranged in a spiral (easy wiring), or a Hilbert curve.

The function is then enabled using `setRemapFunction()`:

```
matrix.setRemapFunction(myRemapFn);
```

## RAM Again

On a per-pixel basis, Adafruit\_NeoMatrix is no more memory-hungry than Adafruit\_NeoPixel, requiring 3 bytes of RAM per pixel. But the number of pixels in a two-dimensional display takes off exponentially...a 16x16 display requires *four times* the memory of an 8x8 display, or about 768 bytes of RAM (nearly half the available space on an Arduino Uno). It can be anywhere from *tricky* to *impossible* to combine large displays with memory-hungry libraries such as SD or ffft.

## Gamma Correction

Because the Adafruit\_GFX library was originally designed for LCDs (having limited color fidelity), it handles colors as 16-bit values (rather than the full 24 bits that NeoPixels are capable of). This is not the big loss it might seem. A quirk of human vision makes bright colors less discernible than dim ones. The Adafruit\_NeoMatrix library uses *gamma correction* to select brightness levels that are visually (though not numerically) equidistant. There are 32 levels for red and blue, 64 levels for green.

The `Color()` function performs the necessary conversion; you don't need to do any math. It accepts 8-bit red, green and blue values, and returns a gamma-corrected 16-bit color that can then be passed to other drawing functions.





# Advanced Coding

---

Help! My Arduino servo code stops working when combined with NeoPixels!

Unfortunately the NeoPixel and Servo libraries don't play nice together; one is dependent on periodically disabling interrupts, the other absolutely requires interrupts. There are a couple of options here:

- Use a dedicated [servo control shield \(http://adafru.it/1411\)](http://adafru.it/1411) or [breakout board \(http://adafru.it/815\)](http://adafru.it/815), offloading that task from the processor so interrupts are a non-issue.
- Use a hardware-PWM-based servo library rather than the stock Arduino Servo library. This can provide rock-steady servo timing without interrupts, but can only control a very limited number of servos (2-3), and only on very specific pins. [PWMServo \(http://adafru.it/aTn\)](http://adafru.it/aTn) appears to handle this...download "Paul's version 2 library" from that page.

How fast can I refresh a string of (N) pixels?

NeoPixels receive data from a fixed-frequency 800 KHz datastream (except for "V1" Flora pixels, which use 400 KHz). One bit therefore requires  $1/800,000$  sec — 1.25 microseconds. One pixel requires 24 bits (8 bits each for red, green blue) — 30 microseconds. After the last pixel's worth of data is issued, the stream must stop for at least 50 microseconds for the new colors to "latch."

For a strip of 100 pixels, that's  $(100 * 30) + 50$ , or 3,050 microseconds.  $1,000,000 / 3,050 = 328$  updates per second, approximately.

## However...

That's only the time needed to push the bits down the wire. The *actual* refresh rate will be something less than this, and can't be estimated as a single number for all cases. It takes time to process each "frame" of animation. How much time depends on the complexity of the math and the efficiency of the code (for example, floating-point calculations can be relatively slow). The technique above gives a maximum *theoretical* rate, but that's just a starting point. Reality in some cases could fall an order of magnitude (or more) below this.

For exploratory benchmarking, you can always write code *as if* a large number of pixels were present, and time the result. The extra output bits will simply be ignored by the strip (or you can even test with no NeoPixels connected at all).

That won't do. Now what?

Because NeoPixels use a fixed-frequency clock, options are limited. You can't switch out for a faster microcontroller and expect substantially different results.

One option is to use a different LED type, such as our LPD8806 strips or WS2801 pixels. These can be driven at higher data rates, though they do have some other tradeoffs with respect to NeoPixels (cost, color resolution and/or pixel density).

Another is to develop your own code for a more capable microcontroller or an FPGA that drives *multiple* NeoPixel strips *in parallel*. One such project — OctoWS2811 for the Teensy 3.0 microcontroller — is shown later. This sort of thing is a complex undertaking and not recommended for beginners. And even among more experienced programmers, there's often an unreasonable over-emphasis on data rates when the *real* bottlenecks lie elsewhere...don't dwell on this too much unless you can confirm it's a problem.

Can I control NeoPixels using (Board X)?

We currently only offer an Arduino library. See the links later for other devices. For anything beyond this, if considering writing your own library, understand that some devices are better suited to the task than others. Read through the timing requirements shown below and determine if the processor or device in question can synthesize a signal meeting those specifications. An 8 MHz AVR can just barely keep up...anything slower may have trouble, though some hardware-specific hacks (like clever use of SPI) might make it possible. In many cases, assembly language is required.

Why not Raspberry Pi?

The Raspberry Pi running Linux is a multitasking system, and control may switch among multiple running programs at any time. As such, it's impossible to guarantee the strict 800 KHz signal required by NeoPixels. You may be able to fudge it for short intervals, but it's not something that can be counted upon. This is why we use LPD8806 pixels for the [Raspberry Pi light painting \(http://adafru.it/aPk\)](http://adafru.it/aPk) demonstration.

## Third-Party Libraries

NeoPixel-compatible libraries have been developed for devices beyond Arduino. Please keep in mind that Adafruit did not develop any of this code, does not know it inside and out, and can't fix bugs or offer technical help. This is Wild West stuff.

- **OctoWS2811 (<http://adafru.it/cDM>)**: specifically for the PJRC Teensy 3.0 microcontroller board. Uses DMA to drive up to 8 NeoPixel strips concurrently with minimal processor load. Multiple boards can be cascaded for still larger displays.
- **FadeCandy (<http://adafru.it/cDN>)**: also for Teensy 3.0. Doesn't support as many pixels as OctoWS2811, but adds smooth interpolation and other features for the color-persnickety.
- **LEDscape (<http://adafru.it/cDO>)**: specifically for BeagleBone Black. Although the BeagleBone is a multitasking Linux system like the not-NeoPixel-compatible Raspberry Pi, this code exploits hardware features specific to the BeagleBone Black to drive *hundreds* of meters of NeoPixel strip with virtually no processor load.
- **WS2812 LED Driver (<http://adafru.it/cDP>)** for Parallax Propeller.
- **xCORE NeoPixel test code (<http://adafru.it/dcO>)** for the XMOS xCORE startKIT.

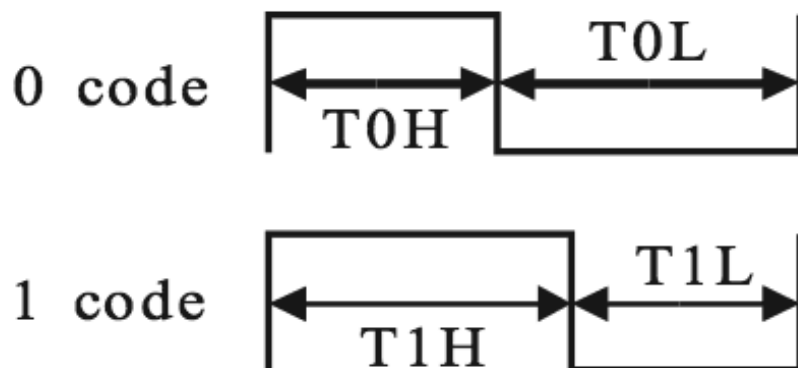
Some of these are 3.3V devices. See the "Powering NeoPixel" page for notes on controlling 5V NeoPixels from 3.3V microcontrollers.

WS2811? WS2812? Why do I see two different names mentioned?

The WS2811 is an earlier driver chip separate from the RGB LED. The data signal is similar, but runs at half the speed. By the time the WS2812 (with integrated LED) was released, a lot of code and projects had already built up around the WS2811 name. Sometimes code “for the WS2811” might actually be for the newer chip, or for either type. The Adafruit\_NeoPixel library supports both.

## Writing Your Own Library

The [WS2812 datasheet \(http://adafru.it/cDB\)](http://adafru.it/cDB) explains the data transmission protocol. This is a *self-clocking* signal — there’s only one wire, not separate data and clock lines. “1” and “0” bits are indicated by varying the duty cycle of a fixed-frequency square wave.



There’s a math goof in the datasheet’s timing values. Use these figures instead:

**Data transfer time** ( $T_H + T_L = 1.25\mu s \pm 300ns$ )

T0H	0 code ,high voltage time	0.4us	±150ns
T1H	1 code ,high voltage time	0.8us	±150ns
T0L	0 code , low voltage time	0.85us	±150ns
T1L	1 code ,low voltage time	0.45us	±150ns
RES	low voltage time	Above 50μs	

Note that there’s nearly 25% “wobble room” in the timing. So if your code can’t match the recommended times exactly, it’s usually okay.

There are three bytes of data for each pixel. These should be issued in green, red, blue order, with the most-significant bit first.

**Composition of 24bit data:**

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

The data for pixel #0 (nearest the microcontroller) is issued first, then pixel #1, and so forth to the furthest pixel. This does not operate like a traditional shift register!

After all the color data is sent, the data line must be held low for a minimum of 50 microseconds for the new colors to “latch.”

You may want to dig through our [Arduino library \(http://adafru.it/aZU\)](http://adafru.it/aZU) for insights. The timing-critical parts are written in AVR assembly language, but it’s extensively commented with C-like pseudocode.

My Microcontroller Isn’t Fast Enough to Do That

The WS2812 appears to be backwardly-compatible with the 400 KHz WS2811 signal. If you can precisely match the latter chip’s timing, either type will respond. **The WS2811 protocol is not simply a half-speed WS2812.** The duty cycle for the “0” and “1” bits is slightly different. From the [WS2811 datasheet \(http://adafru.it/cDS\)](http://adafru.it/cDS):

T0H	0 code,high voltage time	0.5 $\mu$ s	$\pm 150$ ns
T1H	1 code,high voltage time	1.2 $\mu$ s	$\pm 150$ ns
T0L	0 code,low voltage time	2.0 $\mu$ s	$\pm 150$ ns
T1L	1 code,low voltage time	1.3 $\mu$ s	$\pm 150$ ns